International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Web Based Institute Complain Management System

Abhirag Bakode[1], Puja Gupta[2], Aryan Shrivastava[3], Abhishek Goutam[4], Neeraj Kumar Rathore[5]

[1,2,3,4]Dept. of Information Technology, Shri G.S. Institute of Technology and Science, Indore (M.P.)
[5]Professor, NiTTTR, Bhopal

Abstract

This paper presents a web-based Complaint Management System designed for institutional maintenance in colleges and sim-ilar organizations. The system enables faculty members to report maintenance issues, administrators to assign workers and track progress, and maintenance workers to update job status throughout the repair lifecycle. A key differentiator is the *dual-confirmation workflow*: both the assigned worker and the reporting faculty member must independently confirm task completion before a complaint is closed. The system is built using Next.js 15 for the frontend and API layer, MongoDB as the database, and NextAuth.js for authentication with role-based access control. Deployment is handled through Vercel with MongoDB Atlas. This paper covers the system's motivation, architecture, database design, authentication mechanism, implementation details, API structure, security considerations, and directions for future enhancement.

Keywords: complaint management, Next.js, MongoDB, role-based access control, institutional maintenance, full-stack develop-ment, NextAuth.js, dual-confirmation workflow

1 Introduction

1.1 Motivation

Maintenance management in large college campuses is a persistent operational challenge. When infrastructure issues arise—a non-functional tubelight, a leaking tap, a broken door handle—the resolution process is often informal and unreli-able. Complaints are logged in physical registers, communi-cated verbally, or sent via email, with no systematic tracking of progress or accountability.

Existing commercial solutions such as Zendesk, Freshdesk, and Jira Service Management are designed for enterprise-scale customer support operations. They offer features like SLA au-tomation, AI-powered ticket routing, and CRM integrations—capabilities that are both unnecessary and prohibitively expen-sive for a college maintenance context. This project addresses the need for a lightweight, purpose-built system that handles the specific workflow of institutional maintenance complaint resolution.

1.2 Problem Statement

The core challenge is not the filing of complaints but the lack of visibility into what happens afterward. Without a central-ized digital system: complaints are lost or duplicated; workers are not notified of assignments in a timely manner; there is no record of task assignments or completion history; and there is no verification mechanism to ensure reported issues are sat-isfactorily resolved. Faculty members

frequently re-file com-plaints because they receive no status updates, while adminis-trators lack the data to balance workloads across maintenance staff.

1.3 Objectives

The system was designed to achieve the following objectives:

1. Provide role-specific dashboards for administrators, fac-ulty, and workers.
2. Implement a complaint lifecycle that progresses through submission, assignment, progress tracking, and a two-way sign-off for closure.
3. Enforce secure authentication with hashed passwords and JWT-based sessions.
4. Enable PDF report generation for resolved complaints.
5. Support SMS notifications for worker assignment via Twilio.
6. Deploy the application on a publicly accessible platform.

1.4 Scope and Limitations

The system is designed for single-institution deployment with-out multi-tenancy support. It does not currently support im-age/file uploads for documenting issues, real-time chat be-tween stakeholders, integration with existing ERP or college management systems, or real-time push notifications via Web-Sockets.

2 Literature Review

2.1 Commercial and Academic Solutions

Several student projects have addressed similar problems us-ing various technology stacks—MERN implementations, Java Servlet/JSP systems, Flutter applications backed by Google Sheets, and Firebase-powered React apps. While these handle basic CRUD operations and status tracking, very few imple-ment a verification step where the complaint originator con-firms satisfactory resolution. This gap motivated the dual-confirmation design central to our system.

2.2 Role-Based Access Control

Role-Based Access Control (RBAC) is a well-established au-thorization paradigm, formalized by Sandhu et al. [1]. Rather than assigning permissions to individual users, permissions are mapped to roles, and users are assigned roles. In the Next.js ecosystem, RBAC is implemented through middleware that validates JWT tokens before granting route access, combined with server-side authorization checks in API handlers. Nex-tAuth.js provides native support for embedding role informa-tion in session tokens.

3 System Architecture

3.1 Technology Stack

The technology choices were driven by development effi-ciency, cost, and ecosystem compatibility. Table 1 summarizes the stack.

Table 1: Technology Stack

Layer	Choice	Rationale
Framework	Next.js 15	Unified API
		routes, SSR, and
		file-based routing
UI	React 19, Tailwind, DaisyUI	Rapid prototyping

		with pre-built
		components
Database	MongoDB (Mongoose 8)	Flexible schema,
		free Atlas tier
Auth	NextAuth.js 4.24	Native Next.js
		integration, JWT
		+ session support
Passwords	bcryptjs	Industry-standard
		hashing (10 salt
		rounds)
PDF	jsPDF	Client-side
		generation, no
		server overhead
Hosting	Vercel	Free tier, GitHub
		CI/CD, Next.js
		optimized

A monolithic architecture was chosen deliberately—the frontend and backend coexist within the same Next.js project. For this project's scale, separating into distinct frontend and backend services would introduce unnecessary complexity without meaningful benefit.

3.2 Database Schema

3.2.1 User Model

Each user document contains: name, unique email, bcrypt-hashed password, role (one of admin, faculty, worker), and optional department and phone fields. Mongoose manages createdAt and updatedAt timestamps automatically.

3.2.2 Complaint Model

The complaint document references the filing faculty and the assigned worker via MongoDB ObjectIds. Complaint details are stored as an array of description objects. The status field cycles through four states: pending → assigned → in-progress → fulfilled.

Two boolean flags—completedWorker and completedFaculty—implement the dual-confirmation logic. A Mongoose pre-save hook checks both flags on every save operation; when both are true, the status transitions to fulfilled automatically. This centralizes the business logic at the data layer rather than distributing it across API handlers.

Additional date fields track every lifecycle stage: worker visit, material request, material issuance, work completion, faculty confirmation, and fulfillment.

3.2.3 Worker Notifications

A supplementary Worker model includes an embedded noti-fications array and a phone number field for Twilio integra-tion. Each notification carries a message, type, timestamp, and read/unread flag.

3.3 Authentication Architecture

Authentication follows a two-layer design:

Layer 1 — Middleware: A Next.js middleware func-tion intercepts requests to protected routes (/admin/*,

/faculty/*, /worker/*), reads the JWT, validates the
role, and redirects unauthorized users.

Layer 2 — API Route Checks: Each API handler indepen-dently calls getServerSession to verify authentication and authorization. This second layer protects against direct API calls (via curl, Postman, etc.) that bypass the middleware.

The credentials provider in NextAuth.js connects to Mon-goDB, validates the password via bcrypt.compare, and issues a JWT containing the user's ID, name, email, role, de-partment, and phone number. The token is stored as an HTTP-only cookie.

4 Implementation

4.1 Registration and Login

The registration form collects name, email, password, and role selection (faculty or worker), with optional department and phone fields. Passwords are hashed with bcrypt before database insertion. Email uniqueness is enforced at both the application and database levels. Upon successful login, the client-side home page reads the role from the session and redi-rects to the appropriate dashboard.

4.2 Faculty Interface

The faculty dashboard displays filed complaints in *Pend-ing* and *Completed* tabs, with each complaint card showing department, location, description excerpt, and status badge. The complaint submission form includes a department drop-down (16 engineering departments), location field, description textarea, and mobile number input.

When a worker marks a job complete, the faculty member reviews the worker's remark and materials used, provides feedback, and confirms completion—setting the completedFaculty flag and triggering the automatic sta-tus transition if the worker's flag is already set.

4.3 Admin Dashboard

The admin interface features: (a) summary stat cards showing total, idle, and assigned worker counts via MongoDB aggre-gation pipelines; (b) a sortable, searchable complaints table with status filtering; (c) complaint detail pages for worker as-signment, material issuance, timeline viewing, and PDF report generation; and (d) a worker status modal displaying a card grid of all workers labeled as Busy or Free with current as-signment details.

4.4 Worker Interface

The worker dashboard mirrors the faculty layout with pend-ing/completed filters. Workers can request materials (recorded with timestamps for admin action) and mark jobs as completed by submitting work remarks. The interface was designed for simplicity—large buttons, clear labels, and minimal naviga-tion steps.

5 Complaint Lifecycle

The complaint lifecycle follows a structured five-stage pro-gression:

1. **Submission:** Faculty files complaint; status = pending, both completion flags = false.
2. **Assignment:** Admin assigns a worker; status → assigned; SMS notification is triggered.
3. **In Progress:** Worker visits the site, may request materi-als from admin; status → in-progress upon material request or issuance.
4. **Worker Completion:** Worker submits remark and marks complete; completedWorker → true.
5. **Faculty Confirmation:** Faculty verifies the repair quality, provides feedback, and confirms;

completedFaculty → true. The pre-save hook detects both flags and sets status → fulfilled.
The dual-confirmation requirement ensures that complaints are not prematurely closed. A worker marking a job complete does not guarantee the faculty member is satisfied with the outcome—the verification step adds an accountability layer absent from most comparable systems.

6 API Design

The backend API is organized as Next.js route handlers under /app/api/. Key endpoints are summarized in Table 2.

Table 2: Primary API Endpoints

Method	Route	Function
POST	/api/auth/register	User registration
POST	/api/complaints	Submit complaint
GET	/api/complaints/admin	All complaints
		(filterable)
PUT	/api/complaints/[id]	Assign worker / issue
		materials
PUT	/api/complaints/worker/[id]	Worker marks
		complete
PUT	/api/complaints/faculty/[id]	Faculty confirms
		completion
GET	/api/admin/worker-stats	Worker availability
		counts
GET	/api/admin/worker-status	Per-worker status
		details

database connection), the Mongoose connection is cached on the Node.js global object, persisting across warm function in-vocations.

9 Results and Discussion

9.1 Strengths

The dual-confirmation workflow proved to be the system's strongest feature, fundamentally changing the accountability dynamic of complaint resolution. The admin dashboard's real-time worker availability statistics (powered by MongoDB aggregation pipelines with $lookup joins) provided opera-tionally useful insights for workload distribution. Centraliz-ing the status transition logic in the Mongoose pre-save hook avoided scattered business logic across multiple API routes.

9.2 Limitations

Routes are organized by role (/complaints/admin/, /complaints/faculty/, /complaints/worker/), which introduces some URL redundancy but simplifies role-specific authorization logic within each handler.

10. Security

Passwords are hashed with bcrypt (10 salt rounds)—computationally sufficient to make brute-force attacks imprac-tical without degrading login performance. Sessions use JWTs signed with a server-side secret stored in environment vari-ables. The JWT carries role information, eliminating the need for database lookups on every request.
The dual-layer authorization (middleware + API-level checks) ensures that even direct API calls cannot

bypass role restrictions. Input validation is implemented across three lay-ers: HTML5 form validation on the client, Mongoose schema constraints (required fields, enum values, unique indexes), and manual checks in API handlers.

All sensitive configuration—MongoDB connection strings, NextAuth secrets, Twilio credentials—resides in environment variables excluded from version control via .gitignore.

Known security gaps include: the public registration form allows any role selection (including admin), there is no email verification during registration, and JWTs cannot be individu-ally revoked without changing the signing secret.

11. Deployment

The application is deployed on Vercel with automatic CI/CD from GitHub. Each push to the main branch triggers a build and deploy cycle. SSL certificates and CDN distribution are handled automatically. API routes execute as serverless func-tions.

MongoDB Atlas hosts the database on its free tier. To address connection pooling in the serverless environment (where each function invocation could potentially open a new The Twilio SMS notification integration, while functional in local development, encountered credential issues in produc-tion and is currently disabled—meaning workers must manu-ally check the dashboard for new assignments. The registra-tion system lacks access controls for admin account creation. Other gaps include: no image uploads for visual documenta-tion of issues, no real-time updates (page refresh required), no password reset functionality, no analytics or trend reporting, and a hardcoded department list.

12. Future Work

Priority enhancements include:

1. restoring the notification system via Twilio or an email alternative such as SendGrid;
2. adding image upload support using Cloudinary or AWS S3;
3. implementing real-time updates via WebSockets or Server-Sent Events;
4. building an analytics dashboard for complaint trends, resolution times, and worker efficiency;
5. introducing priority levels with SLA timers and escalation rules;
6. devel-oping a mobile companion application using React Native or Flutter; and
7. securing the registration process with admin-only account creation and email domain restrictions.

13. Conclusion

This paper presented a Complaint Management System that addresses the specific workflow needs of institutional mainte-nance operations. While the underlying problem is fundamen-tally a CRUD application with role-based routing, the imple-mentation details—particularly the dual-confirmation work-flow, material tracking integration, and real-time worker avail-ability monitoring—distinguish it from typical ticketing sys-tems. Next.js proved to be an effective choice for this class of application, providing a unified framework for frontend ren-dering, API logic, and server-side operations. MongoDB's flexible schema accommodated an evolving data model dur-ing development. The system demonstrates that modern JavaScript tooling enables a small team to build and deploy a functional institutional tool without significant infrastructure investment.

References

1. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
2. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
3. Gupta, P. and Kulkarni, N., 2013. An introduction of soft computing approach over hard computing. International Journal of Latest Trends in Engineering and Technology (IJLTET), 3(1), pp.254-258
4. Gupta, P., Shukla, M., Arya, N., Singh, U. and Mishra, K., 2022. Let the Blind See: An AIIoT-Based Device for Real-Time Object Recognition with the Voice Conversion. In Machine Learning for Critical Internet of Medical Things (pp. 177-198). Springer, Cham.
5. Gupta, P., Kulkarni, A. and Sarda, A., 2013. An embedded health care supervisory systems. International Journal of Latest Trends in Engineering and Technology (IJLTET), 3, pp.379-386
6. Kushwaha, U., Gupta, P., Airen, S. and Kuliha, M., 2022, December. Analysis of CNN Model with Traditional Approach and Cloud AI based AppGupta, P., Sharma, V. and Varma, S., 2022. A novel algorithm for mask detection and recognizing actions of human. Expert Systems with Applications, p.116823.roach. In 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS) (pp. 835-842). IEEE.
7. Gupta, P., Varma, S., Arya, N. and Bhagel, R., 2023. Intelligent Security System Based on the Internet of Things (IoT). In Intelligent Sensor Node-Based Systems (pp. 177-191). Apple Academic Press
8. Rathore, P., Gupta, P., Jain, S. and Shrivastava, Y., 2022. A Study of the Automated Vehicle Number Plate Recognition System. i-manager's Journal on Pattern Recognition, 9(2), p.30.
9. Rajput, A., Gupta, P., Ghodeshwar, P., Varma, S., Sharma, K.K. and Singh, U., 2023, June. Study of Cloud Providers (Azure, Amazon, and Oracle) According To Service Availability and Price. In 2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN) (pp. 1177-1188). IEEE.
10. Gupta, P., Arya, N., Singar, C.P., Chaudhari, A., Singh, U. and Gupta, S., 2025. Safety of Pedestrians in AI-Optimized VANETs for Autonomous Vehicles via Real-Time Vehicle-to-Vehicle Communication. In AI-Driven Transportation Systems: Real-Time Applications and Related Technologies (pp. 169-181). Cham: Springer Nature Switzerland.
11. Gupta, P.,Singh, U., Shukla, M., Sharma, V., Varma, S. and Sharma, S.K., 2023. Acknowledgment of patient in sense behaviors using bidirectional ConvLSTM. Concurrency and Computation: Practice and Experience, 35(28), p.e7819.
12. Gupta, P.,Singh, U., and Shukla, M., 2022. Activity detection and counting people using Mask-RCNN with bidirectional ConvLSTM. Journal of Intelligent & Fuzzy Systems, 43(5), pp.6505-6520.